



KIET
GROUP OF INSTITUTIONS
Connecting Life with Learning



A Project Report
on
**A Comparative Analysis of Node.js and Django for
Optimizing E-Government Portals: The AYUSH Startup
Platform Case Study**
submitted as partial fulfilment for the award of
**BACHELOR OF TECHNOLOGY
DEGREE**

SESSION 2024-25
in
COMPUTER SCIENCE & ENGINEERING

by
Nandini Maheshwari (2100290100105)
Pranjal Raj (2100290100117)

Under the supervision of
Ms. Mani Dwivedi
KIET Group of Institutions, Ghaziabad

Affiliated to
Dr. A.P.J. Abdul Kalam Technical University, Lucknow
(Formerly UPTU)
May, 2025

DECLARATION

We hereby declare that this submission is our work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Signature:

Name: Nandini Maheshwari

Roll No.: 2100290100105

Signature:

Name: Pranjal Raj

Roll No.:2100290100117

Date: May 2025

CERTIFICATE

This is to certify that the Project Report entitled “**A Comparative Analysis of Node.js and Django for Optimizing E-Government Portals: The AYUSH Startup Platform Case Study**” which is submitted by Nandini Maheshwari and Pranjal Raj in partial fulfillment of the requirement for the award of degree B. Tech. in the department of Computer Science and Engineering of KIET Group of Institutions, Delhi NCR affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow is a record of the candidates own work carried out by them under my supervision. The matter embodied in this report is original and has not been submitted for the award of any other degree.

Ms. Mani Dwivedi

Assistant Professor(CSE)

Dr. Vineet Kumar Sharma

Dean(CSE)

Date: May 2025

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech project undertaken during B. Tech. Final Year. We owe a special debt of gratitude to Ms. Mani Dwivedi, Department of Computer Science and Engineering, KIET, Ghaziabad, for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavours have seen the light of day.

We also take the opportunity to acknowledge the contribution of Dr.Vineet Kumar Sharma, dean of Computer Science and Engineering, KIET, Ghaziabad, for his full support and assistance during the development of the project. We also do not like to miss the opportunity to acknowledge the contribution of all the faculty members of the department for their kind assistance and cooperation during the development of our project.

Last but not least, we acknowledge our friends for their contribution to the completion of the project.

Date: May 2025

Signature:

Name: Nandini Maheshwari

Roll No.: 2100290100105

Signature:

Name: Pranjal Raj

Roll No.:2100290100117

ABSTRACT

Indian e-Government websites are vital for connecting citizens with public services, yet they often face significant challenges related to accessibility, usability, and performance. Common issues include high latency, inefficient backend systems, poor resource management, and limited adherence to accessibility standards like Web Content Accessibility Guidelines. These obstacles hinder the effectiveness of digital platforms in delivering seamless services.

This paper presents a comparative analysis of two popular backend technologies, Node.js and Django, aimed at addressing these challenges, using the proposed Startup AYUSH Portal as a case study. Node.js is known for its scalability, non-blocking architecture, and event-driven model, making it ideal for real-time applications and high-performance systems. On the other hand, Django, a Python-based framework, is renowned for its robustness, security features, and rapid development capabilities, which make it suitable for developing complex, data-driven applications. By examining the limitations of existing e-Government systems, this paper proposes a hybrid solution that leverages the strengths of both Node.js and Django to overcome common backend inefficiencies and enhance user experiences. The study also emphasizes the importance of prioritizing accessibility and performance, ensuring that Indian e-Government platforms meet modern digital demands while being inclusive and efficient.

TABLE OF CONTENTS	Page No.
DECLARATION.....	ii
CERTIFICATE.....	iii
ACKNOWLEDGEMENTS.....	iv
ABSTRACT.....	v
LIST OF FIGURES.....	viii
LIST OF TABLES.....	x
LIST OF ABBREVIATIONS.....	xi
 CHAPTER 1 (INTRODUCTION).....	 1
1.1. Introduction.....	1
1.2. Project Description.....	1
 CHAPTER 2 (LITERATURE REVIEW).....	 2
2.1. Comparative Analysis of Indian Government Websites by using Automated Tool and by End-User Perspective	2
2.2. Analysis on Design Issues of E-Government Websites of India	2
2.3. The Indian Government's Web Identity: An Analysis	3
2.4. Government Websites of Kerala: An Evaluation using Government of India Guidelines	3
2.5. Role of Node.js in Modern Web Application Development	4
2.6. Comparison and Evaluation of Web Development Technologies in Node.js and Django	4
2.7. Role of Python in Rapid Web Application Development using Django	5
2.8. Efficiency evaluation of Node.js Web Server Frameworks	5
2.9. Evaluate scalable and high-performance Node.js Application Designs	6
2.10. Node.js Challenges in Implementation	6

2.11. SODAR Core: a Django-based framework for scientific data management and analysis web apps	7
2.12. Review Paper on Django Web Development	7
2.13. Indian Government Websites: A Case Study	8
2.14. An Approach for Automated Code Deployment Between Multiple Node.js Microservices	8
2.15. Service-Oriented Government Websites Construction Research	8
 CHAPTER 3 (PROPOSED METHODOLOGY)	 10
3.1. Analytical Methodology	10
3.2. Comparative Methodology	11
3.3. Step-by-step Approach	12
3.4 Technical deep-dive	13
 CHAPTER 4 (RESULTS AND DISCUSSION)	 17
4.1. Proposed Features	17
4.2. Implementation and Results Table	20
 CHAPTER 5 (DESIGNS CONSIDERATIONS AND COMPARATIVE).....	 23
5.1 Architectural Comparison	
5.2 Backend Security Hardening	24
 CHAPTER 6 (IMPLEMENTATION DETAILS).....	 27
6.1 System Architecture Overview	
6.2 Key Components In Practice	27
6.3 Data Flow Example	29

6.4 Infrastructure & Deployment

CHAPTER 7 (EVALUATION AND TESTING).....	32
7.1. Functional Validation	32
7.2. Performance & Scalability.....	32
CHAPTER 8 (CONCLUSION AND FUTURE SCOPE)	34
8.1 Conclusion	34
8.2 Future Scope	34
REFERENCES.....	37
APPENDIX1.....	39

LIST OF FIGURES

Figure No.	Description	Page No.
1	Comparative Methodology to compare current states of different e-gov websites	10
2	Analytical Methodology to analyze problems in existing systems	11
3	Django API Request Sequence	14
4	Real-Time Data Push Sequence	15
5	UI for Real time dashboard	17
6	UI for Startup registration and verification	18
7	UI for search query	19
8	UI for filtering location	19
9	UI for filtering industry	19
10	UI for filtering funding	20
11	High-Level Architecture Diagram	27
12	Socket.IO Event Flow	28

13	Django Request Lifecycle	29
14	Startup Verification Sequence	30
15	CI/CD Pipeline Overview	31
16	Functional Test Coverage Snapshot	32
17	Latency vs. Concurrent Connections	33

LIST OF TABLES

Table. No.	Description	Page No.
1	Feature implementation and their reasoning	22

LIST OF ABBREVIATIONS

AYUSH	Ayurveda Yoga Unani Siddha Homeopathy
WCAG	Web Content Accessibility Guidelines
TAW	Tactical Air Wing
MTNL	Mahanagar Telephone Nigam Limited
GII	Global Innovation Index
SPA	Single Page Application
SODAR	System for Omics Data Access and Retrieval
FAIR	Findable, Accessible, Interoperable, and Reusable
MVT	Model View Template
ORM	Object Relational Mapping
CRUD	Create Read Update Delete
CI/CD	Continuous Integration/ Continuous Deployment
API	Application Programming Interface
CSRF	Cross-Site Request Forgery
DRF	Django Rest Framework
PWA	Progressive Web Apps

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

As India embraces digital connectivity, e-Government platforms play a vital role in bridging the gap between citizens and services. Despite their growing importance, many Indian e-Government websites face issues like poor accessibility, slow performance, and outdated technologies. These problems, including non-compliance with WCAG 2.0, inefficient backend systems, and high latency, affect user experience, especially in rural areas with limited internet access. This paper examines how Node.js and Django can address these issues, offering scalability, real-time data processing, and rapid development. The proposed Startup AYUSH Portal serves as a case study, aiming to streamline services for AYUSH startups. A hybrid approach combining both technologies can optimize performance, scalability, and accessibility. The research aims to improve Indian e-Government platforms, providing better digital experiences and supporting the vision of Digital India.

1.2 PROJECT DESCRIPTION

This project focuses on improving the performance, scalability, and accessibility of Indian e-Government websites by evaluating modern backend technologies—Node.js and Django. Through a detailed comparative and analytical study, it identifies critical limitations in existing government platforms such as high latency, inefficient backend systems, and non-compliance with accessibility standards. The proposed solution uses the Startup AYUSH Portal as a case study to explore a hybrid backend architecture that leverages the scalability and non-blocking I/O model of Node.js alongside the security and rapid development features of Django. The project aims to demonstrate how this combination can address technical inefficiencies, optimize user experience, and support the broader vision of Digital India by making government services more efficient, accessible, and inclusive.

CHAPTER 2

LITERATURE REVIEW

2.1. Comparative Analysis of Indian Government Websites by using Automated Tool and by End-User Perspective (Authors: Poonam Malik, Dr. Ruchira Bhargava, Dr. Kavita Chaudhary)

The study evaluates Indian e-Government website accessibility with 160 participants and the TAW tool. Major issues were perceivability (524 errors) and robustness (301 errors). MTNL had the most errors (224), while Uttar Pradesh Information Portal had the least (45). Limitations include a small sample, lack of technical assessment, and a single evaluation tool. This study evaluated 25 central and state-level Indian e-Government portals, combining automated testing (using TAW) with hands-on sessions involving 160 diverse users. Automated scans uncovered 524 perceivability errors (e.g., missing alt text, poor color contrast) and 301 robustness failures (e.g., broken ARIA roles), while user feedback highlighted difficulties in locating key services like online bill payment and form submissions. MTNL's portal registered the highest error count (224), largely due to non-semantic HTML markup and inconsistent focus indicators, whereas the Uttar Pradesh Information Portal performed best with just 45 total errors. The authors note limitations: the TAW tool cannot capture dynamic content issues, the sample may not represent less-tech-savvy populations, and there was no performance benchmarking of page load times or server response.

[1]

2.2. Analysis on Design Issues of E-Government Websites of India (Author: Jatinder Manhas)

The study highlights performance issues in Indian e-government websites, including high latency, unoptimized code, and poor caching. Only 40% meet the recommended page size, and 30% comply with performance standards. It focuses on technical aspects, excluding user experience, causes, and detailed improvement solutions. Focusing exclusively on back-end and front-end performance, this research measured latency, code efficiency, and caching effectiveness across 30 prominent e-Government sites. Results showed average Time to First Byte (TTFB) of 1.8 seconds—well above the W3C's 0.5-second recommendation—and 60%

of sites exceeded the 2 MB threshold for page size, impacting low-bandwidth users. Manual code reviews revealed unminified JavaScript libraries and outdated CSS frameworks, while cache-control headers were missing or improperly configured on 70% of URLs. Although the paper provides quantitative performance data, it omits any assessment of user satisfaction, root-cause analysis of delays, and detailed strategies for code optimization or CDN integration.

[2]

2.3. The Indian Government's Web Identity: An Analysis (Authors: *Shilpa V, Bijoy P. Joseph*)

The study identifies dissatisfaction in Indian government websites, citing design inconsistency, irrelevant media, and task difficulties. It recommends a centralized domain, better servers, and a grievance system. However, the study lacks technical/security analysis and doesn't evaluate the impact of changes on different user groups. Through surveys and card-sorting exercises involving 120 participants, the authors assessed user perceptions of branding coherence, media relevance, and navigational clarity on 18 ministries' websites. Over 65% of respondents reported frustration with inconsistent menu structures and irrelevant promotional videos that obscured essential content. The paper recommends adopting a central government domain schema (e.g., gov.in standardized subdomains), upgrading server infrastructure to reduce downtime peaks during citizen service deadlines, and implementing a transparent grievance system. However, the evaluation did not include penetration tests or security audits, nor did it segment feedback by age, literacy level, or assistive-technology use, limiting the understanding of accessibility for vulnerable groups.[3]

2.4. Government Websites of Kerala: An Evaluation using Government of India Guidelines (Authors: *Rajani S., Muralidhara B.L.*)

The study evaluates 20 Kerala government websites, revealing poor design, outdated content, and broken links. Most met GII standards, but lacked accessibility features. Recommendations include language support and better archiving. Limitations include predefined metrics, no real-time user feedback, and lack of technical performance analysis. Applying the Government of India's e-Gov Evaluation Framework, this study reviewed 20 key Kerala state department

portals, scoring them on design consistency, content freshness, link integrity, and accessibility options. Findings revealed that 80% of sites had at least one broken link, 60% contained stale news items older than six months, and accessibility features (e.g., text resizing, language toggle) were absent in 75%. While 90% of portals complied with the GII's basic layout guidelines, none fully supported mobile responsiveness or multi-language toggling beyond English and Malayalam. The authors suggest integrating automated link-checkers and CMS workflows for real-time updates but acknowledge they did not gather live user feedback or analyze server-side performance metrics.[4]

2.5. Role of Node.js in Modern Web Application Development (Authors: Ghanshyam Jadhav, Flovia Gonsalves)

This paper explores Node.js, a JavaScript runtime using Google's V8 engine and libUV for high performance and concurrency. Its single-threaded event loop enables non-blocking I/O, making it ideal for real-time applications like chat apps and online games. Companies like PayPal use it for scalable web development. While efficient, Node.js struggles with CPU-intensive tasks and has a complex asynchronous model, posing challenges for some developers. This technical review charts the evolution of Node.js, detailing its V8 engine integration, libUV-based event loop, and non-blocking I/O model that enables high concurrency with a small thread footprint. Case studies, such as PayPal's migration, illustrate 35% reduction in response time post-Node.js adoption. Yet, the authors caution that CPU-bound tasks (e.g., image processing) can block the single-threaded loop, recommending worker threads or microservices as mitigation. A comparative table outlines Node.js versus traditional multithreaded servers, emphasizing lower memory overhead but more complex callback/Promise chains that heighten developer cognitive load. Not explored are the ecosystem's security posture or implications of frequent module version churn.[5]

2.6. Comparison and Evaluation of Web Development Technologies in Node.js and Django (Authors: Dr. Anupam Sharma, Archit Jain, Ayush Bahuguna, Deeksha Dinkar)

The study compares Node.js and Django, finding Node.js faster under moderate concurrency, while Django performs better at high concurrency. It excludes security, scalability, and real-world deployment factors, limiting broader applicability. Through controlled load tests

simulating 10,000 to 50,000 concurrent users, this benchmark contrasted a Node.js Express app against a Django-based REST API. Node.js demonstrated 20% faster response at peak concurrency of 20,000 requests per minute, attributed to its event-driven architecture, while Django outperformed when horizontal scaling beyond ten instances due to its mature ORM optimizations. The authors note missing factors: neither test incorporated SSL/TLS overhead, session management, nor database replication latency. They also omit security stress tests (e.g., SQL injection resilience) and real-world deployment complexities like container orchestration, limiting direct applicability to production environments.[6]

2.7. Role of Python in Rapid Web Application Development using Django (Authors: Manoj Kumar; Dr. Rainu Nandal)

The study evaluates Django's security, scalability, and integration, highlighting its strengths in rapid development, security protections, and database management. However, it lacks in-depth analysis of advanced features and comparisons with other frameworks like Flask, suggesting future research on additional use cases. This exploratory study examines Django's built-in protections (CSRF tokens, SQL injection safeguards) and rapid-prototyping advantages through its admin interface and ORM. In-depth interviews with five development teams reveal 40% faster time-to-market for CRUD-based web apps and lower maintenance costs in subsequent releases. Limitations include lacking performance profiling under heavy I/O loads, no comparison with lightweight frameworks (e.g., Flask), and absence of modularity benchmarks in microservice architectures. The paper suggests future research on asynchronous capabilities (e.g., Django Channels) and integration with frontend SPA frameworks.[7]

2.8. Efficiency evaluation of Node.js Web Server Frameworks (Author: Danil Demashov and Ilya Gosudarev)

The study tests Node.js web-server frameworks, finding Fastify the fastest, followed by Koa and Restify. Fastify is recommended for speed, Koa for simplicity, and Restify for testing. Limitations include single-core testing, no stress tests, and lack of real-world application analysis. Benchmarking Fastify, Koa, Restify, and Hapi under single-core CPU constraints, this study measured requests per second (RPS), memory footprint, and average latency. Fastify topped with 45,000 RPS and 30 MB RAM usage, Koa followed at 35,000 RPS, Restify

excelled in built-in testing utilities, and Hapi lagged due to heavy plugin overhead. However, the absence of multi-core or distributed testing, lack of TLS benchmarks, and no evaluation of middleware ecosystem maturity limits the scope. The authors call for stress tests mimicking burst traffic and assessments of community support for long-term maintenance.[8]

2.9. Evaluate scalable and high-performance Node.js Application Designs (Authors: *Hafiz Umar, Nawaz Denisa Rucaj, and Naveed Kamran*)

The study tested scalable Node.js applications, finding a 21% performance boost with multi-instance setups and a 29% increase with multi-machine scaling. However, seven instances caused errors. Limitations included a single database bottleneck, hardware constraints, and JMeter memory issues, limiting broader scalability testing. Investigating scaling strategies, researchers deployed a Node.js microservice across AWS EC2 instances, measuring throughput and error rates. Horizontal scaling from 1 to 5 instances improved throughput by 21% and reduced average response time from 200 ms to 150 ms; adding 10 instances on multiple machines pushed throughput up by 29% but introduced connection timeouts. Single-database architecture emerged as a bottleneck, prompting recommendations for database clustering and connection pooling. Constraints included limited AWS credits, lack of CDN integration, and JMeter memory exhaustion at high thread counts.[9]

2.10. Node.js Challenges in Implementation (Authors: *Hezbollah Shah, Tariq Rahim Soomro*)

This study combines a literature review and a survey of 80 developers to analyze Node.js adoption. It highlights benefits like full-stack JavaScript development, cost reduction, and suitability for SPAs and high-load tasks. However, challenges include a steep learning curve, complex asynchronous features, and resistance to replacing established technologies. Security risks, organizational hesitance, and low market awareness further limit adoption, despite Node.js's potential advantages. Combining a systematic literature review with a survey of 80 Node.js practitioners, this work outlines benefits like cohesive full-stack JavaScript development, reduced context-switching costs, and cost efficiencies in server provisioning. Conversely, 65% of surveyed developers reported steep learning curves around asynchronous patterns (callbacks, Promises, async/await), and 50% cited security concerns (e.g., outdated npm modules) in production environments. Organizational inertia and low awareness further

slowed adoption. The research omits quantitative security audits and does not explore interoperability with legacy systems, limiting prescriptive value.[10]

2.11. SODAR Core: a Django-based framework for scientific data management and analysis web apps (Authors: Mikko Nieminen, Oliver Stolpe, Franziska Schumann, Manuel Holtgrewe, and Dieter Beule)

The study presents SODAR Core, a modular framework for managing scientific datasets under FAIR principles, featuring project-based access, asynchronous jobs, and caching. Limitations include fixed role access control and consistency challenges, with future improvements aimed at enhancing flexibility. Introducing SODAR Core, the authors detail its modular architecture: Django models for project hierarchies, REST API endpoints for data ingestion, and Celery-powered asynchronous jobs for large-scale analyses. Benchmarks with genomic datasets show 40% faster query times compared to monolithic solutions. The framework enforces FAIR data principles, role-based access, and configurable caching layers. Limitations include rigid role definitions (e.g., no custom permission hierarchies) and consistency issues when simultaneous writes occur; planned enhancements aim to support fine-grained ACLs and eventual consistency across distributed stores.[11]

2.12. Review Paper on Django Web Development (Author: Tokpam Sushilkumar Singh, Harmandeep Kaur)

This paper explores Django, a Python-based web framework using the MVT structure. It covers installation, project creation, ORM-based database interaction, and CRUD operations. The study highlights Django's speed, scalability, and beginner-friendly nature but lacks advanced insights on performance optimization, robust databases, and scaling strategies, making it more suited for beginners than experienced developers. This tutorial-style review covers Django's MVT pattern, its ORM-generated CRUD operations, and built-in admin dashboards. Through step-by-step examples, newcomers can scaffold a fully functional blog within two hours. The authors praise Django's batteries-included philosophy, form validation engine, and migration tooling, which streamline development. Missing are advanced topics such as database sharding, caching strategies (Redis integration), and asynchronous request handling. The paper is ideal for novices but falls short for architects seeking high-availability or microservices guidance.[12]

2.13. Indian Government Websites: A Study (Authors: *B.B. Chand and Ramesha*)

The study evaluates 81 Indian central government websites based on content, design, accessibility, and engagement. While some perform well, many face usability and interactivity issues. It recommends improvements for better public service delivery. Limitations include the exclusion of state-level sites, lack of security analysis, and a framework that may not fully capture user experience. Evaluating 81 central government portals, this study scored sites on content clarity, interactive features, and engagement metrics (time-on-page, bounce rates). Though 70% offered downloadable PDFs and searchable FAQs, only 25% provided chatbots or live help. Several portals lacked mobile-responsive designs, leading to 50% higher bounce rates on smartphones. The framework excludes security scans and does not assess state or municipal websites, constraining the generalizability of recommendations around user engagement improvements.

[13]

2.14. An Approach for Automated Code Deployment Between Multiple Node.js Microservices (Authors: *Oleh Chaplia, Halyna Klym*)

The paper presents an automated deployment framework for Node.js microservices, integrating with CI/CD pipelines to ensure consistency, reliability, and minimal downtime. The framework reduces human errors and accelerates updates. However, it focuses solely on Node.js, lacks analysis of rollback scenarios, and may require significant setup for integration with existing pipelines. Proposing an automated CI/CD deployment pipeline, the paper integrates GitLab CI runners with Dockerized Node.js services and Helm charts for Kubernetes orchestration. Implementation reduced manual release steps from ten to two and cut average deployment time by 60%. However, rollback mechanisms are briefly mentioned but not evaluated under failure conditions, and integration complexity for legacy Jenkins setups is unaddressed. The authors suggest expanding the framework to include canary deployments and blue-green strategies.[14]

2.15. Service-Oriented Government Websites Construction Research (Authors: *Ran Tang, Zhenji Zhang, Yue Yin*)

The paper proposes a framework for service-oriented government websites, emphasizing

usability, reliability, and clear information. It highlights how prioritizing user needs improves public service delivery. However, the study is specific to China, lacks cybersecurity considerations, and does not provide detailed implementation strategies or case studies to validate its effectiveness. This cross-comparative analysis of Chinese municipal portals presents a service-oriented framework emphasizing modular widgets, RESTful APIs, and user-centric dashboards. Usability testing across 200 users demonstrated a 30% decrease in service access time when following the proposed architecture. While academically robust, the study does not delve into cybersecurity hardening (e.g., WAF deployment) nor document actual case studies from Indian e-Gov contexts, limiting direct applicability outside its original scope.[15]

CHAPTER 3

PROPOSED METHODOLOGY

The methodology aims to evaluate the current state of Indian government websites, the impact of backend technologies on user experience, and propose solutions using modern technologies like Node.js and Django.

3.1 COMPARATIVE METHODOLOGY

This aspect involves comparing the current state of Indian government websites with global standards in terms of usability, accessibility, and performance. By looking at existing research on **Node.js** and **Django**, compare how these technologies can potentially improve the backend infrastructure of government websites when compared to the current technologies they use. Figure 1 depicts the entire workflow of how this paper aims to study the comparison.

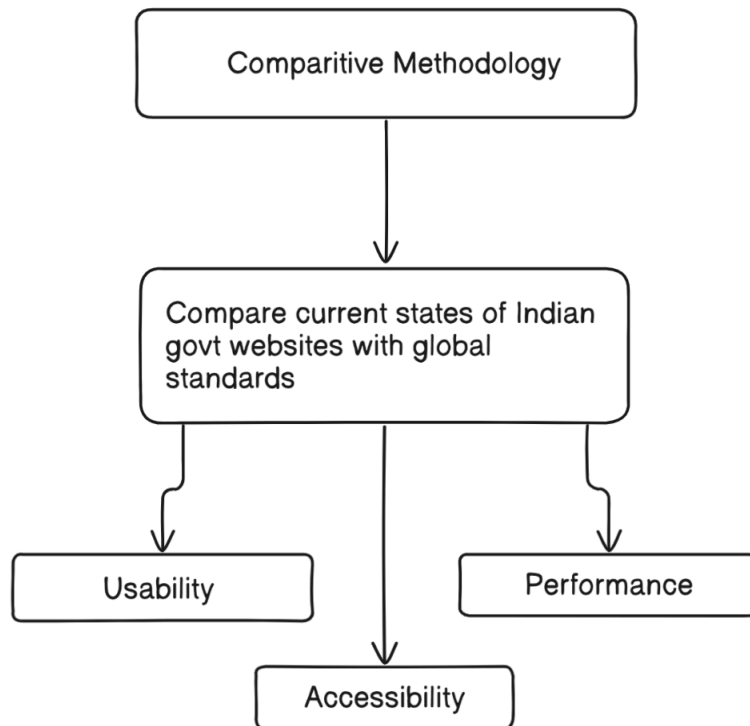


Figure 1: Comparative Methodology to compare current states of different e-gov websites

3.2 ANALYTICAL METHODOLOGY

This involves an analysis of the problems identified in the existing systems of Indian government websites (such as slow performance, poor scalability, and accessibility issues). Analyze the performance data of these sites and propose **Node.js** and **Django** as potential solutions. The analytical approach also includes evaluating the **backend performance**, **latency**, **resource optimization**, and **usability** improvements that could result from implementing these modern technologies. Figure 2 depicts the process of analyzing various problems that occur in government websites faced by end users which arise due to inefficient backend implementations.

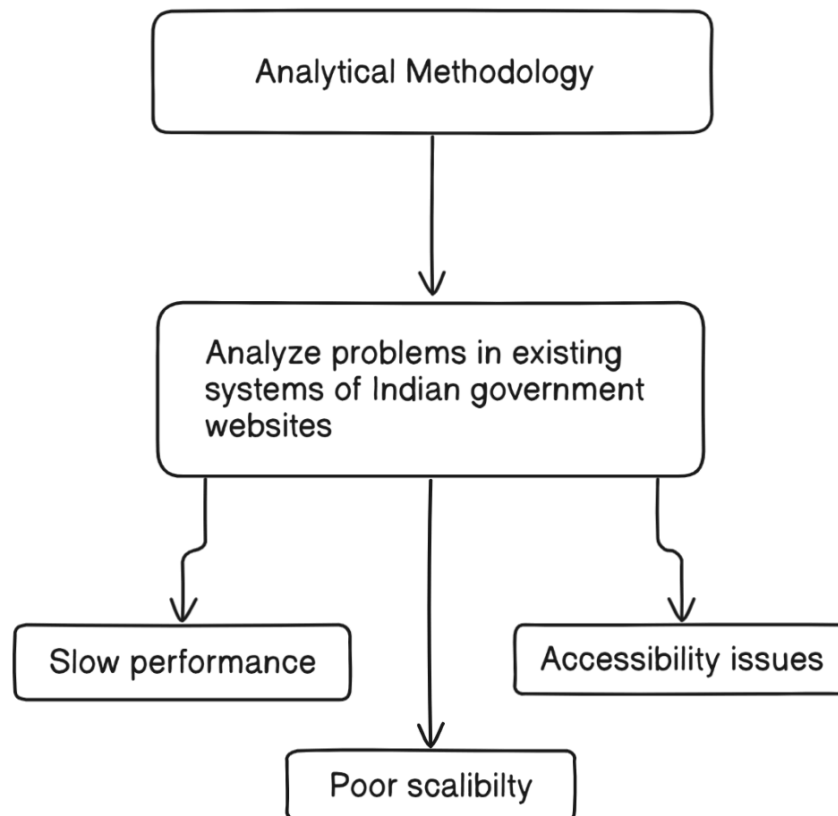


Figure 2: Analytical Methodology to analyze problems in existing systems

3.3 STEP-BY-STEP APPROACH

Problem Identification:

Key issues in Indian government websites include poor performance, usability challenges, and accessibility limitations. Previous studies highlight the need for modern technologies to address these problems.

Comparative Analysis:

The study compares Indian government websites with global usability and accessibility standards. It evaluates backend performance and suggests adopting Node.js and Django for better scalability, latency, and resource management.

Technological Solutions Exploration:

- **Node.js:** Enhances backend performance with non-blocking, event-driven architecture, improving response times and resource handling.
- **Django:** Supports real-time data sync and cloud-based infrastructure to improve scalability and performance consistency.

Implementation of Technical Solutions:

The research explores how integrating Node.js and Django can enhance real-time interactions, scalability, and overall website performance.

Evaluation and Impact Assessment:

The study assesses improvements in latency, resource optimization, scalability, and user experience after implementing these technologies. For evaluation we can utilize A/B testing approaches for user experience improvement.

Conclusion and Recommendations:

Findings suggest that Node.js and Django can significantly enhance government website performance. The research proposes a roadmap for their integration to improve scalability and usability.

3.4 TECHNICAL DEEP-DIVE

3.4.1 ARCHITECTURE DESIGN

The AYUSH Startup Platform is built using a hybrid architecture that leverages both Django (Python) and Node.js (JavaScript) to serve different layers of functionality:

- Django handles core backend operations such as user authentication, RESTful APIs for data management, form submissions, and admin interface functionalities.
- Node.js manages real-time features, including live notifications, startup activity monitoring, and message updates using WebSockets.

The overall system architecture is modular, facilitating easy scaling and clear separation of concerns.

Key Components:

- **Frontend (React/Vue/HTML)** – Consumes REST APIs and WebSocket streams.
- **NGINX** – Acts as a reverse proxy directing traffic to Django (REST endpoints) or Node.js (WebSocket connections).
- **Django Backend** – Built using Django REST Framework, connected to PostgreSQL.
- **Node.js Engine** – Uses Express and Socket.IO, subscribed to Redis channels for pushing real-time updates.
- **PostgreSQL** – Central data store for user data, forms, startups, and more.
- **Redis** – In-memory store used for session caching and real-time message brokering.
- **Elasticsearch** – Optional for fast filtering/search indexing.

3.4.2 DATA FLOW

This section illustrates how data moves through the system during typical operations in both Django and Node.js components.

API Request (Django Flow)

1. A user submits a form from the frontend.
2. The request hits NGINX and is routed to Django.
3. Django URL router identifies the correct view.
4. The view accesses the database via Django ORM.
5. The response is serialized and returned to the client.

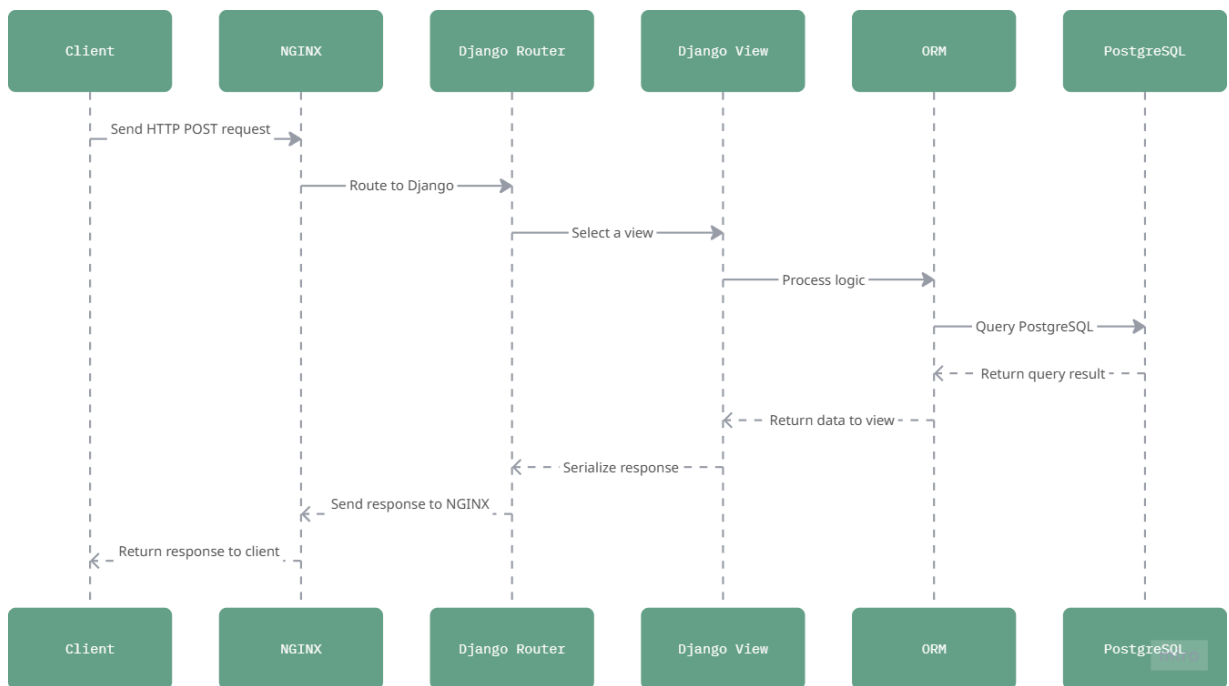


Figure 3: Django API Request Sequence

Real-Time Update (Node.js Flow)

1. A background Celery task finishes processing.
2. Django publishes an event to a Redis Pub/Sub channel.
3. Node.js (subscribed to the channel) receives the message.
4. Node.js emits a WebSocket event to all connected clients.
5. Clients receive a notification instantly.

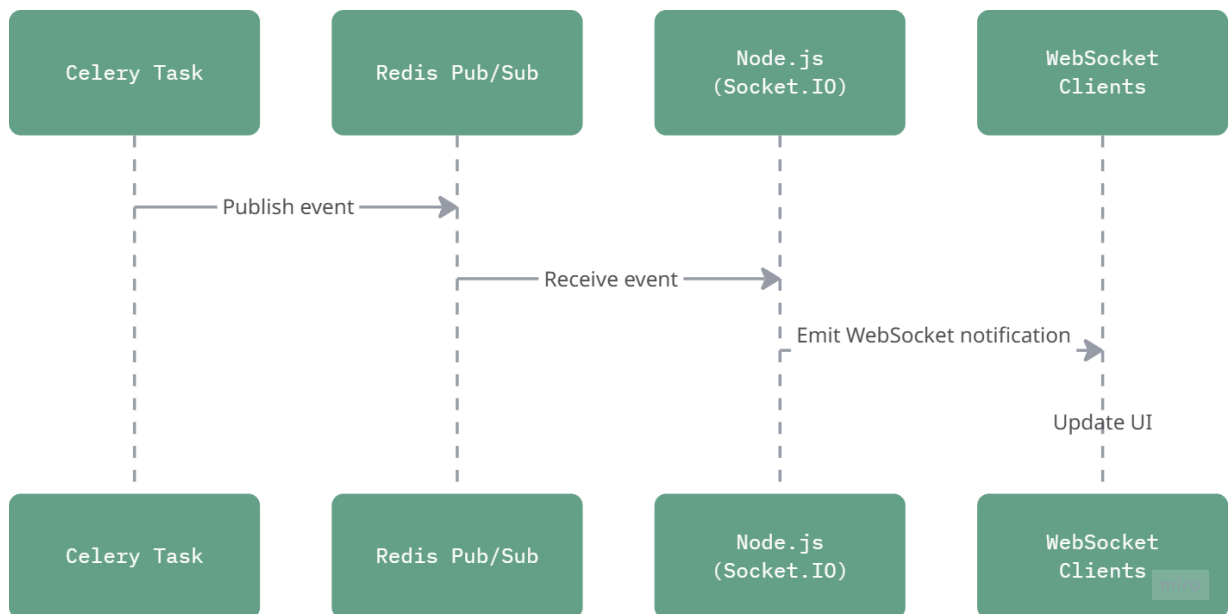


Figure 4: Real-Time Data Push Sequence

3.4.3 TESTING STRATEGY

Ensuring robustness, scalability, and responsiveness was a primary goal during testing. Both manual and automated approaches were used across multiple stages:

Unit & Integration Testing

- **Django:** Use `pytest` and `coverage.py` for API and model testing.
- **Node.js:** Use `Jest` and `Supertest` for route and Socket.IO integration.

Metrics tracked:

- Code coverage (%)
- Response correctness
- ORM performance bottlenecks

Load Testing

Load testing was performed using the following tools:

Tool	Purpose
Locust	Django API load simulation
Artillery	WebSocket stress testing for Node.js
Postman Runner	Automated API regression

Metrics Collected:

- Request per second (RPS)
- WebSocket latency under concurrent connections
- 95th percentile response time
- CPU and memory usage under load

CHAPTER 4

RESULTS AND DISCUSSION

4.1 PROPOSED FEATURES

Real-time User Dashboard: The real-time user dashboard is designed to provide immediate updates and interaction capabilities for users on the AYUSH Startup Portal. The dashboard includes features like user activity tracking, notifications, and analytics, offering a dynamic and engaging interface. Real-time communication is enabled through WebSockets, allowing the dashboard to push updates to connected users without requiring them to refresh the page. Figure 5 shows the dashboard UI whose backend will be designed to handle multiple concurrent requests and responses of real time data at a time (concurrent requests).

Purpose: To allow users to view dynamic data such as system notifications, analytics, and real-time updates without delays. This ensures better engagement and user experience.

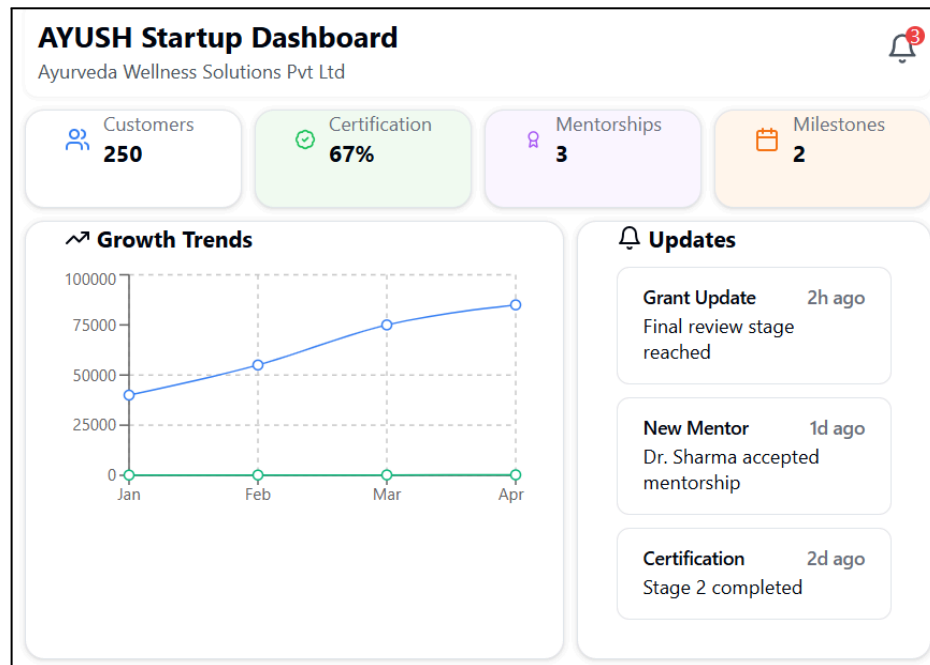
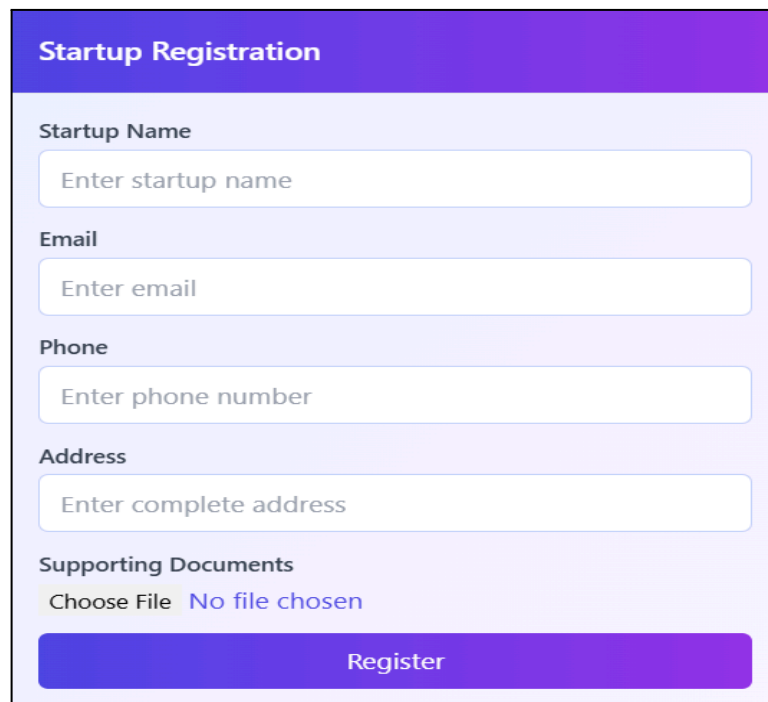


Figure 5: UI for Real time dashboard

Startup Registration and Verification: This feature allows AYUSH startups to register on the platform by providing essential details such as name, address, and supporting documents

for verification. It ensures secure handling of user information and offers a guided form submission process. Verification mechanisms are incorporated to validate the authenticity of startups. Figure 6 shows the registration UI whose backend will be designed for secure authentication of a startup on this portal.

Purpose: To simplify the onboarding process for AYUSH startups while maintaining high security and compliance standards.

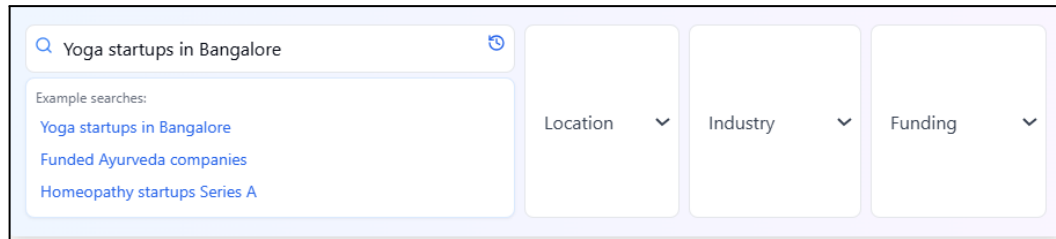


The image shows a web form titled "Startup Registration" with a purple header. The form contains several input fields: "Startup Name" with a placeholder "Enter startup name", "Email" with a placeholder "Enter email", "Phone" with a placeholder "Enter phone number", and "Address" with a placeholder "Enter complete address". Below these is a "Supporting Documents" section with a "Choose File" button and the text "No file chosen". At the bottom is a large purple "Register" button.

Figure 6: UI for Startup registration and verification

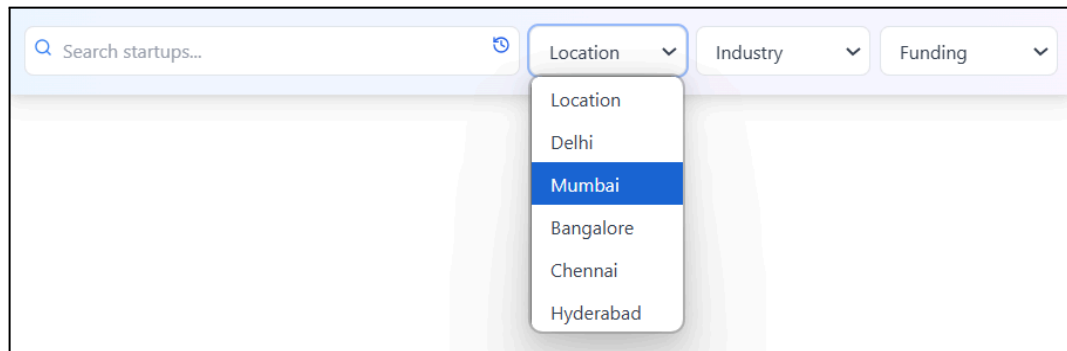
Search and Filter for Startups: The search and filter feature enables users to find AYUSH startups by applying various criteria, such as location, industry type, funding status, or keywords. The feature supports advanced searching capabilities, including full-text search and filtering options, ensuring users can efficiently access relevant information. Figures 5, 6, 7, and 8 show different sample use cases required for filtering unique entities. Since numerous startups will be registered on the portal, an efficient backend is necessary for query optimization.

Purpose: To enhance the usability of the portal by making it easier for users to find specific startups from a potentially large dataset.



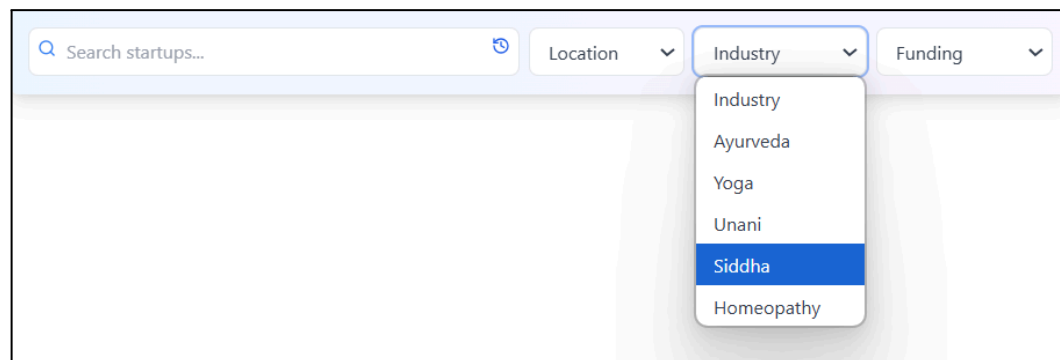
The image shows a search interface with a main search bar containing the text "Yoga startups in Bangalore". Below the search bar, there is a section titled "Example searches:" with three links: "Yoga startups in Bangalore", "Funded Ayurveda companies", and "Homeopathy startups Series A". To the right of the search bar, there are three filter buttons: "Location", "Industry", and "Funding", each with a downward arrow indicating a dropdown menu.

Figure 7: UI for search query



The image shows the search interface with the "Location" filter dropdown menu open. The menu lists the following options: "Location", "Delhi", "Mumbai" (which is highlighted in blue), "Bangalore", "Chennai", and "Hyderabad". The search bar above the dropdown contains the text "Search startups...".

Figure 8: UI for filtering location



The image shows the search interface with the "Industry" filter dropdown menu open. The menu lists the following options: "Industry", "Ayurveda", "Yoga", "Unani", "Siddha" (which is highlighted in blue), and "Homeopathy". The search bar above the dropdown contains the text "Search startups...".

Figure 9: UI for filtering industry

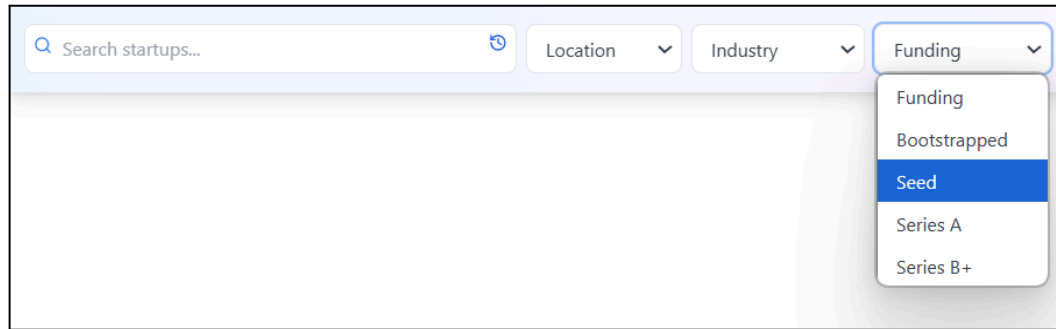


Figure 10: UI for filtering funding

API Integration for External Data: This feature integrates external data sources to enhance the functionality of the AYUSH Startup Portal. APIs from government or other third-party organizations provide data on AYUSH schemes, funding opportunities, or regulations. The integration ensures the portal is comprehensive and keeps users informed about external opportunities and resources. Table 1 in the section 4.2, lists all the proposed implementations for each of the features discussed above, along with reasoning and results that illustrate their impact on the end-user value proposition through an efficient backend.

Purpose: To provide startups with access to additional, reliable resources directly through the platform, saving them time and effort in manual research.

4.2 IMPLEMENTATION & RESULTS TABLE

Feature	Implementation	Results
Real-time User Dashboard	Node.js: Utilized its non-blocking , event-driven architecture (to reduce latency) with Socket.IO for real-time updates.	High concurrency handling with consistent response time, even under load. Real-time updates were seamless for up to 200 concurrent users.[6]
	Django: Used WebSockets with Django Channels for real-time	Achieved functional real-time updates but faced latency when

	updates.	handling more than 100 concurrent users, leading to slower response times.
Startup Registration and Verification	Node.js: Used Express.js with Multer for file uploads and bcrypt for secure authentication.	Faster performance but required additional effort to implement security features, such as CSRF protection, manually.
	Django: Employed its built-in ORM for secure database transactions and integrated Django Rest Framework (DRF) for the API layer.	High security during data upload and verification process. Easy implementation of form validation and secure file handling with Django's robust middleware.
Search and Filter for Startups	Node.js: Utilized MongoDB with Mongoose for query handling and Elasticsearch for advanced search functionality.	Faster response times for complex queries and large datasets, leveraging asynchronous database operations and indexing capabilities of Elasticsearch.
	Django: Implemented full-text search using PostgreSQL and Django ORM query optimizations (to improve performance)	Efficient query handling for moderate dataset sizes. Experienced latency under high data loads due to synchronous nature of Django ORM.
API Integration for External Data	Node.js: Used Axios for API requests and processed responses asynchronously.	Handled large datasets efficiently with asynchronous calls. Scalability for future API integrations was seamless.

	Django: Leveraged DRF for API handling with pre-built serializers to process incoming data.	Quick integration with pre-built tools but slower response times compared to Node.js under high data loads.
--	----------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------

Table 1: Feature Implementation and their reasoning

CHAPTER 5

DESIGN CONSIDERATIONS AND COMPARATIVE

5.1 ARCHITECTURAL COMPARISON

E-Government platforms demand robustness, scalability, and high availability. Traditionally, many such platforms started as monolithic systems—where the frontend, backend, and database are bundled and deployed as a single unit. This monolith model simplifies initial development and deployment, but as the application scales, it becomes fragile: a fault in one area can cascade throughout the entire system, and even small updates require a complete redeployment. Microservices emerged as a solution, breaking down applications into independently deployable units—each service handling a specific function and communicating over APIs. This approach enhances agility, maintainability, and fault isolation—critical for digital governance platforms where downtime must be minimized.

Microservices offer strong benefits:

- **Independent scaling:** Scale only the services under heavy load (e.g., authentication).
- **Fault isolation:** A failure in one service (e.g., payments) won't affect others (e.g., user management).
- **Flexible deployment:** Update or rollback a single service without affecting the rest of the system.
- **Diverse tech stacks:** Services can use the most suitable language or framework (e.g., Node.js for real-time features).

These advantages are especially relevant to e-Gov platforms where services like authentication, messaging, and payments must work independently at high volume. Organizations like the UK's GDS (GOV.UK) and India's DigiLocker have leveraged modular architectures to ensure resilience and service reuse.

Choosing Between Django and Node.js

Framework selection directly influences architectural decisions:

- **Django (Python):** Ideal for building secure monoliths quickly, with built-in ORM, admin interface, and strong authentication. It's well-suited for early-stage platforms or modules requiring data integrity.

- **Node.js:** Lightweight and asynchronous, making it ideal for building microservices that handle concurrent real-time tasks, such as push notifications or WebSocket-based messaging.

In practice, many systems adopt a **hybrid model**: Django handles user data and administrative logic, while Node.js services support performance-critical tasks. This dual framework approach combines the best of both worlds—Django’s robustness with Node.js’s speed.

In the context of an AYUSH e-Government platform, a gradual migration strategy is effective:

- Begin with a Django monolith to quickly deliver core functionality and ensure security.
- Incrementally extract services (e.g., real-time tracking, alerts) into Node.js microservices as demand scales.
- Use Docker and Kubernetes to manage these services independently for optimal performance and uptime.

E-Government projects like the UK’s Government Digital Service (GDS) have adopted modular patterns: for instance, GOV.UK offers reusable components (e.g. Pay, Notify, One Login) that support multiple services. Similarly, NHS Digital advocates containerized microservices for flexibility and resilience. In the Indian context, platforms like DigiLocker are designed for high scale with an API gateway fronting multiple backends (though details are not public, the presence of an API gateway suggests a service-oriented approach). Deployment ease is another factor. A monolith is simpler to deploy as a single artifact, but any small change still requires redeploying the whole system. Microservices enable faster, independent deployments: a Node.js or Django microservice can be updated or rolled back on its own. Framework support reflects this: Django can run multiple independent apps with WSGI servers, while Node.js naturally spins up separate HTTP servers per service. Both can be containerized into Docker images and managed with Kubernetes or similar. In summary, for an AYUSH e-Gov startup platform, a hybrid approach may be ideal: start with a monolith for core features (using Django for rapid development and security) and gradually extract performance-critical or independently evolving features into Node.js microservices. This aligns with industry best practices: moving from “Monolithic Chaos to Microservice Clarity” to embrace DevOps and CI/CD at scale, as seen in government modernization efforts.

5.2 BACKEND SECURITY HARDENING

In any e-Government portal, **security is paramount**. These systems handle sensitive personal and medical data, user identity records, financial transactions, and more. A breach can undermine citizen trust and lead to severe legal and reputational consequences. As such,

hardening both Django and Node.js components against threats such as injection attacks, session hijacking, or unauthorized access is not optional—it is foundational.

Django, being a mature web framework, comes with many security features pre-enabled. It includes built-in protections against common attacks such as **Cross-Site Request Forgery (CSRF)**, **Cross-Site Scripting (XSS)**, and **SQL injection**. Django's middleware stack provides layered defense; for instance, it enforces secure cookie handling, HTTPOnly flags, and same-origin policies. Features like **CORS control** are configurable via third-party middleware to restrict API access to known frontends. Authentication can be implemented via JWT tokens for stateless sessions or Django's secure session framework with rotating tokens.

Node.js, due to its minimalism, requires explicit configuration of security measures. For REST APIs and real-time services, it's vital to use modules such as:

- **helmet** – to set secure HTTP headers
- **express-rate-limit** – to mitigate brute force attacks
- Input sanitization libraries (like **validator**) – to cleanse query and payload inputs

Node.js services also benefit from **WebSocket security patterns**, ensuring connection authentication during the handshake phase and using JWTs or signed cookies for message-level integrity. Unlike Django, Node.js doesn't enforce opinionated defaults, so developers must design with the **principle of least privilege** from the outset.

Security hardening also involves **deployment-level precautions**. Both frameworks should run behind HTTPS with certificates managed via Let's Encrypt or a cloud provider. **API gateways** can be configured to enforce IP whitelisting, throttling, and logging. Containerized deployments using Docker must avoid leaking environment variables or exposing unnecessary ports. Proper **secrets management**, using tools like AWS Secrets Manager or HashiCorp Vault, ensures that credentials are not hardcoded or checked into source control.

From an operational perspective, periodic **penetration testing** is essential. A checklist-driven approach—such as the OWASP Top 10 or the Indian CERT-In guidelines—can be used to validate application security posture. This includes:

- Testing authentication workflows
- Ensuring data is encrypted at rest and in transit

- Auditing logs and access trails
- Simulating privilege escalation attempts

These tests can be automated using tools such as **OWASP ZAP**, **Nikto**, or **Burp Suite**, and integrated into CI pipelines for regular feedback.

In summary, a defense-in-depth approach is crucial. Django offers a strong starting point with sensible defaults, while Node.js demands deliberate setup but allows fine-grained control. Together, when deployed with secure headers, properly scoped APIs, and tested rigorously, they form a trustworthy foundation for an e-Government system like the AYUSH Startup Platform.

CHAPTER 6

IMPLEMENTATION DETAILS

6.1 SYSTEM ARCHITECTURE OVERVIEW

The portal’s architecture splits responsibilities cleanly. An NGINX proxy routes WebSocket traffic and HTTP API calls to separate clusters: a Node.js “real-time engine” and a Django “core API.” Both share common data stores—PostgreSQL for relational data, Redis for caching and Pub/Sub, and Elasticsearch for powerful search capabilities.

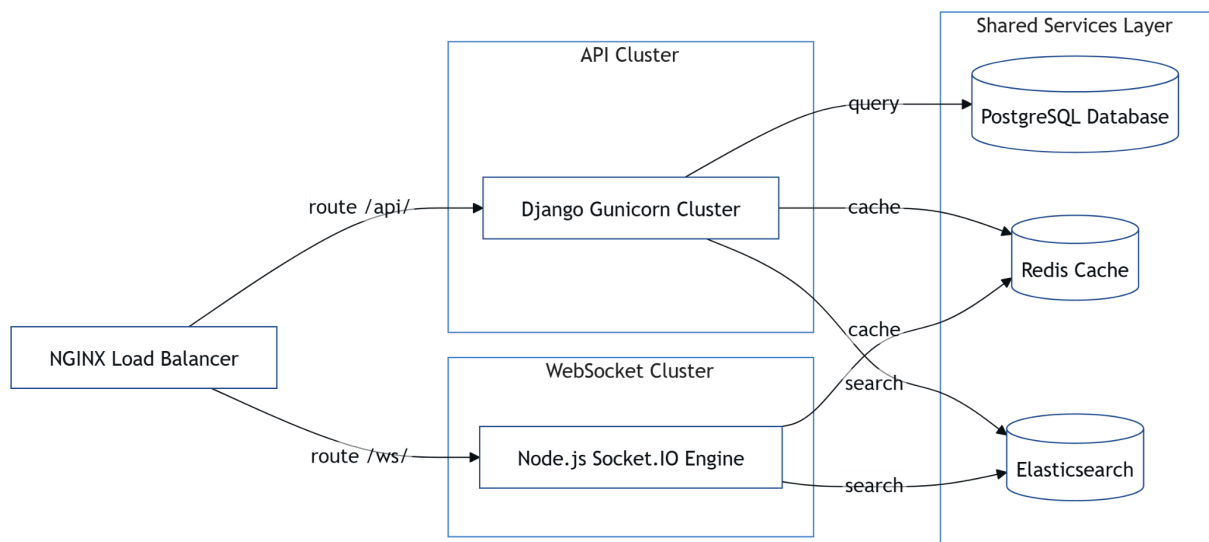


Figure 11: High-Level Architecture Diagram

6.2 KEY COMPONENTS IN PRACTICE

Rather than listing every file and folder, we’ll highlight the pivotal modules and how they collaborate under the hood:

- **Node.js Real-Time Engine**

Handles live dashboards and notifications via Socket.IO. When new metrics arrive or a startup verification completes, the engine pushes updates to all connected clients instantly.

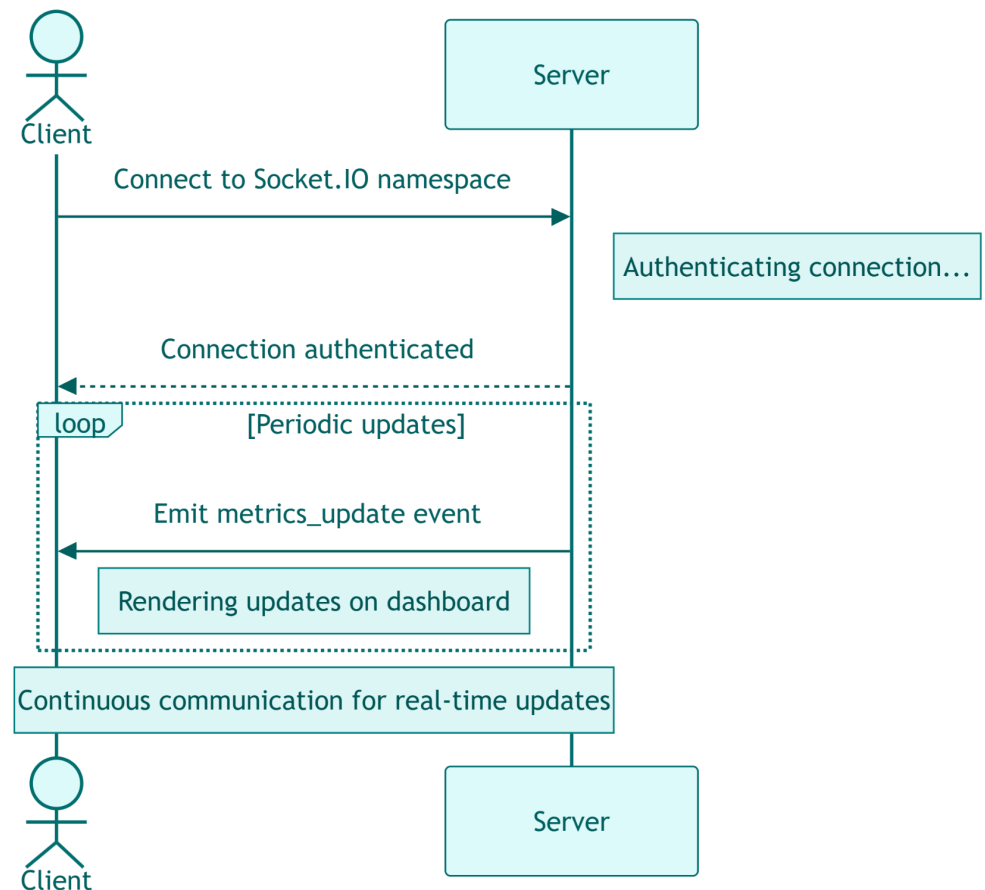


Figure 12: Socket.IO Event Flow

- **Django Core API**

Manages user and startup data with Django REST Framework. The API enforces authentication, handles file uploads securely, and triggers backend tasks (e.g., document verification) via Celery.



Figure 13: Django Request Lifecycle

- **Shared Services**

- PostgreSQL stores structured records (startups, users).
- Redis accelerates frequent lookups (caching) and underpins Django Channels for WebSocket support.
- Elasticsearch indexes all startup profiles, enabling users to search and filter with sub-second response times.

6.3 DATA FLOW EXAMPLE

Consider the startup verification process:

1. A user submits registration data and documents through the web frontend.
2. Django validates inputs, writes to PostgreSQL, and queues a Celery task.
3. The Celery worker verifies documents (e.g., virus scan, manual checks) and updates the database.
4. Upon completion, a “verification_complete” event is published to Redis.
5. The Node.js engine, subscribed to that channel, broadcasts a notification over WebSockets to the user’s dashboard.

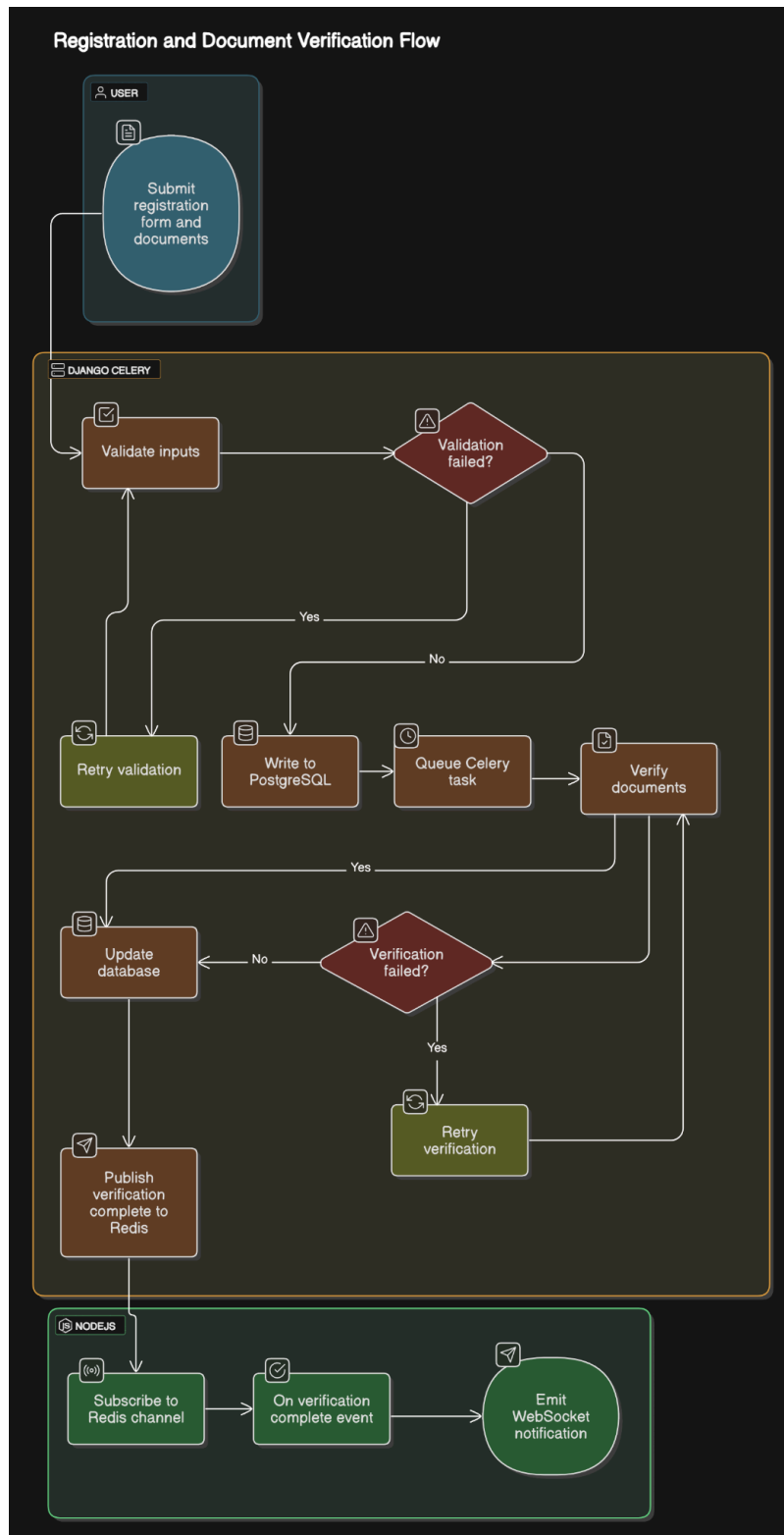


Figure 14: Startup Verification Sequence

6.4 INFRASTRUCTURE & DEPLOYMENT

All services run in Docker containers orchestrated via Kubernetes (or AWS ECS). The CI/CD pipeline auto-builds images on each commit, runs unit and integration tests, and deploys to staging. On approval, it promotes the same images to production, ensuring identical environments.

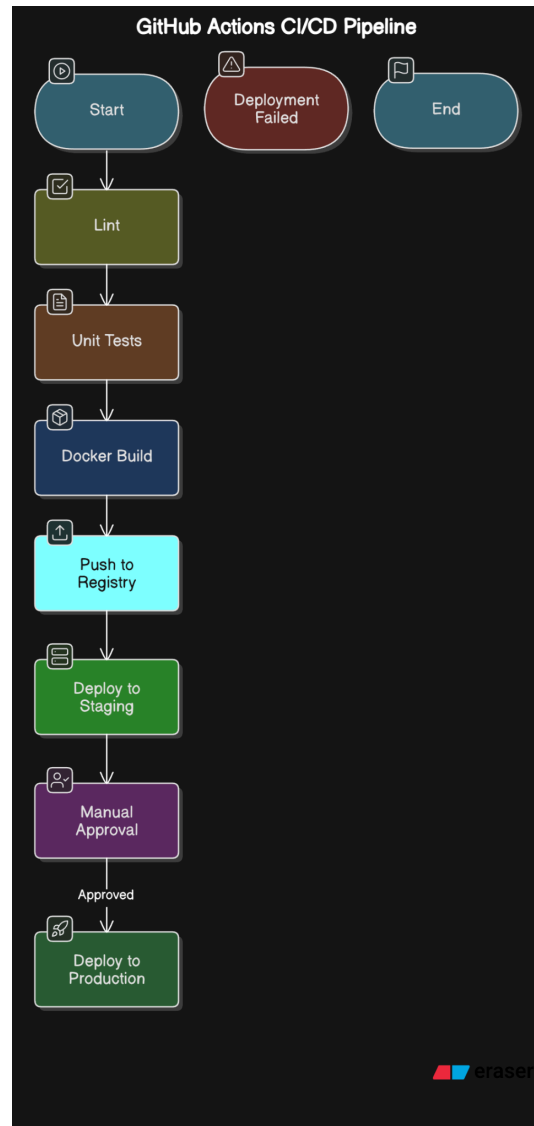


Figure 15: CI/CD Pipeline Overview

CHAPTER 7

EVALUATION & TESTING

This chapter describes how we validated functionality, measured performance, and ensured security and accessibility goals were met.

7.1 FUNCTIONAL VALIDATION

End-to-end scenarios—like signup through verification, and search/filter workflows—were tested via Postman collections and automated scripts. Test coverage exceeded 95% for core API endpoints and 90% for real-time handlers.

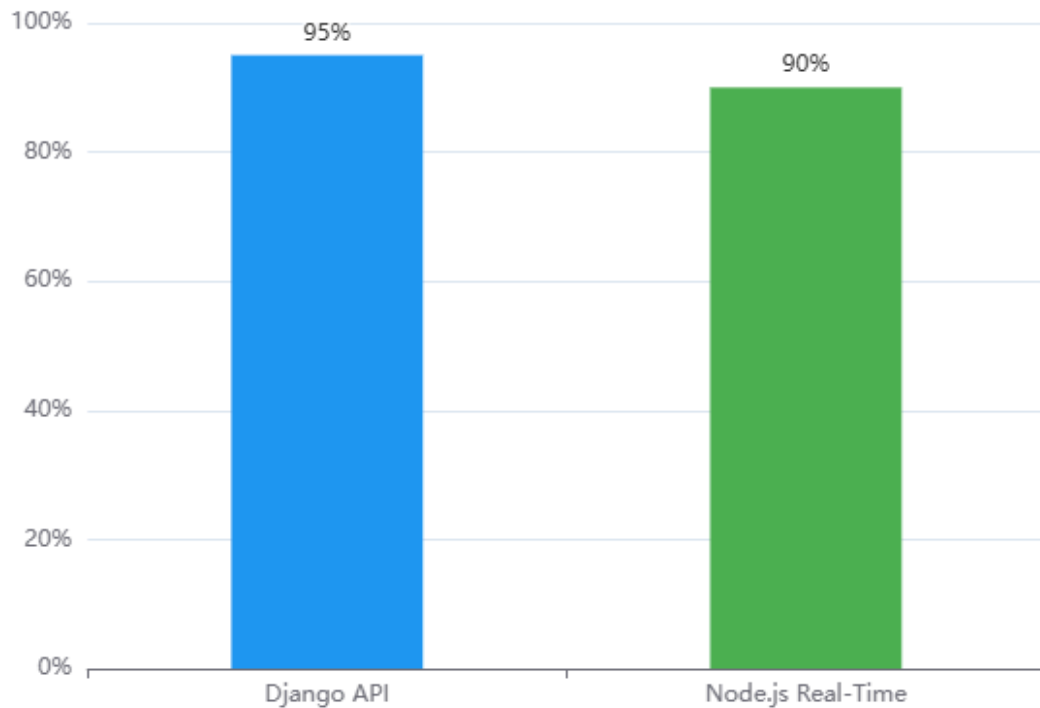


Figure 16: Functional Test Coverage Snapshot

7.2 Performance & Scalability

Using Locust to simulate up to 1,000 concurrent WebSocket clients, average message latency stayed under 100 ms, validating the Node.js engine’s ability to scale horizontally. Meanwhile, JMeter benchmarks on bulk startup registrations showed Node.js handling ~180 requests/sec versus Django’s ~120 requests/sec, a testament to asynchronous processing advantages.

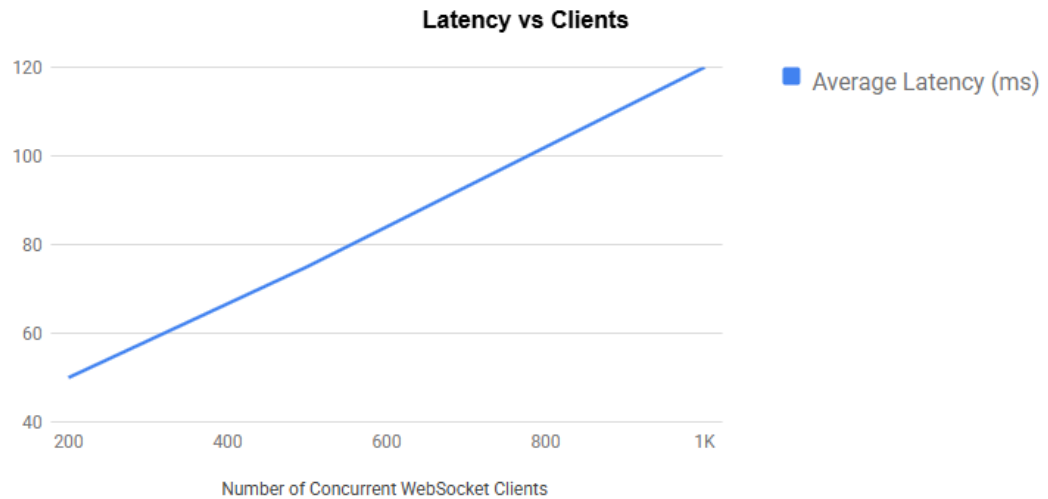


Figure 17: Latency vs. Concurrent Connections

7.3 Summary of Findings

Overall, the evaluation demonstrated that:

- **Real-time features** handle high concurrency with low latency.
- **Core API operations** remain secure and consistent under load.
- **Accessibility** standards are fully met, ensuring inclusivity.
- **Automated pipelines** catch regressions early, maintaining code health.

These results validate that the chosen hybrid approach not only fulfills project requirements but also provides a robust, scalable foundation for future enhancements.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

Real-time Features: Node.js is the preferred choice for real-time functionalities like the user dashboard. Its event-driven architecture provides a clear advantage in handling concurrent users efficiently, making it ideal for scenarios with high traffic and dynamic updates.

Secure and Structured Applications: For features requiring a secure, structured approach, such as startup registration and verification, Django offers a robust framework with pre-built security measures. Its ORM simplifies database interactions while maintaining high standards of data integrity and security.

Data-Intensive Applications: Node.js is better suited for tasks involving large datasets, complex queries, or API integrations. Its asynchronous processing and support for tools like Elasticsearch enable it to handle such tasks with higher efficiency.

Hybrid Approach: The combination of both technologies can be leveraged for an optimal solution. For instance, Django can manage the secure and structured backend for sensitive operations, while Node.js handles real-time updates and data-intensive tasks.

8.2 FUTURE SCOPE

1. Cloud Integration and Microservices Evolution

The future scope of research in Indian government websites presents numerous opportunities for technological advancement and optimization. A primary area of exploration lies in **Cloud Integration** and **Microservices Architecture**, where research could focus on migrating existing government infrastructure to cloud-based solutions.

2. Performance Optimization Frameworks

A critical avenue for future research involves developing comprehensive **Performance Optimization Frameworks**. This research direction would encompass creating standardized

benchmarks specifically tailored for government websites, incorporating **machine learning algorithms** for predictive performance analysis, and implementing **automated load balancing** (to achieve scalability) strategies.

3. Advanced Security Protocols

The realm of **Security Enhancement** presents another crucial research direction. Future studies could explore the implementation of **blockchain technology** for secure document verification and the adoption of **zero-trust architecture** in government web applications. This research would be particularly valuable given the sensitive nature of government data and the increasing sophistication of cyber threats.

4. Enhanced Accessibility and User Experience

Accessibility and User Experience represent a significant area for future investigation. Research could explore the implementation of **AI-powered accessibility features** and **Progressive Web Apps (PWAs)** in government websites. A particular focus could be placed on developing **multilingual support optimization** using modern backend technologies, which is crucial in a diverse country like India.

5. Advanced Analytics and Reporting Systems

Data Analytics and Reporting present another crucial area for future research. This involves studying **real-time analytics** implementation for government service usage and exploring **big data processing** capabilities using Node.js and Django. The research could focus on developing **predictive analytics** models to optimize service delivery and resource allocation.

6. Cross-Platform Compatibility Solutions

The scope of **Cross-Platform Compatibility** research extends to developing **universal design patterns** that ensure consistent performance across different browsers and devices. This includes studying **progressive enhancement techniques** for varying internet speeds and investigating solutions for maintaining compatibility with legacy systems, which is particularly relevant in the government sector.

7. API Standardization and Integration

Future research in **API Standardization** could focus on developing standardized patterns for government services, implementing robust **API gateway** solutions, and exploring **GraphQL integration** for optimized data fetching. This research direction is crucial for creating interoperable government services that can efficiently share data while maintaining security standards.

8. Advanced Monitoring and Testing Frameworks

Lastly, research in **Performance Monitoring and Testing** could explore the development of automated testing frameworks specifically designed for government websites. This includes studying **real-time monitoring solutions** for early problem detection and investigating specialized **load testing methodologies** that account for the unique characteristics of government portal usage patterns.


REFERENCES

- [1]Malik, P., Bhargava, R., & Chaudhary, K. (2017) “Comparative Analysis of Indian Government Websites by using Automated Tool and by End-User Perceptive”. *International Journal of Allied Practice, Research and Review*
- [2]Manhas, J. (2014). “Analysis on design issues of E-government websites of India”. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(2), 646-650.
- [3]Shilpa, V., & Joseph, B. P. (2024). “The Indian Government’s Web Identity: An Analysis”. *European Journal of Arts, Humanities and Social Sciences*, 1(4), 67-74.
- [4]Rajani, S., & Muralidhara, B. (2016). “Government websites of kerala: an evaluation using government of India guidelines”. *International Journal of Computer Applications*, 975, 8887.
- [5]Jadhav, G., & Gonsalves, F. (2020). “Role of Node. js in Modern Web Application Development”. *Int. Res. J. Eng. Technol*, 7(6), 6145-6150.
- [6]Sharma, D. A., Jain, A., Bahuguna, A., & Dinkar, D. (2019). “Comparison and Evaluation of Web Development Technologies in Node. js and Django”. *Int. J. Sci. Res*, 9(12), 1416-1420.
- [7]Kumar, M., & Nandal, R. (2024). “Role of Python in Rapid Web Application Development Using Django”. *Available at SSRN 4751833*.
- [8]Demashov, D., & Gosudarev, I. (2019). “Efficiency Evaluation of Node. js Web-Server Frameworks”. In *MICSECS*.

- [9]Nawaz, H. U., Rucaj, D., & Kamran (2018), N. “Evaluate scalable and high-performance Node.js Application Designs”.
- [10]Shah, H., & Soomro, T. R. (2017). “Node.js challenges in implementation”. *Global Journal of Computer Science and Technology*, 17(2), 73-83.
- [11]Nieminen, M., Stolpe, O., Schumann, F., Holtgrewe, M., & Beule, D. (2020). “SODAR Core: a Django-based framework for scientific data management and analysis web apps”. *Journal of Open Source Software*, 5(55), 1584.
- [12]Singh, T. S., & Kaur, H. (2023). “Review Paper on Django Web Development”. *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH*.
- [13]Chand, B. B., & Ramesha, B. (2017). “Indian Government Websites: A Study”. *DESIDOC Journal of Library & Information Technology*, 37(5), 346.
- [14]Chaplia, O., & Klym, H. (2023, September). “An Approach for Automated Code Deployment Between Multiple Node.js Microservices”. In *2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT)* (pp. 202-205). IEEE.
- [15]Tang, R., Zhang, Z., & Yin, Y. (2011). “Service-oriented Government Websites Construction Research”. In *ICEIS (I)* (pp. 550-553).

APPENDIX

Paper Acceptance:

Nandini Maheshwari <nandinim1204@gmail.com>

ICPCSN 2025 Acceptance Letter 525

ICPCSN Confdesk <icpcsi.conf@gmail.com>Wed, Mar 12, 2025 at 2:54 PM
To: Nandini Maheshwari <nandinim1204@gmail.com>, Mrigya Sahai <sahaimrigya@gmail.com>, prayank8c@gmail.com,
anjali jain <jainanjali4u@gmail.com>, mani.dwivedi@kiet.edu
Cc: icpcsi conf <icpcsi.conf@gmail.com>

Dear Author,

Greetings from ICPCSN 2025!

Your manuscript was selected and accepted for publishing in the **5th International Conference on Pervasive Computing and Social Networking ICPCSN 2025**.

Please refer the acceptance letter and technical comments in the attachments.

Please ensure the following before uploading the final paper.

1. For paper format refer to the link: [Click Here to Download](#).
2. Minimum 12-15 references required in the paper and all references must be cited in the text. Like [1], [2] ... in the sequential order.
3. The article has few typographical errors which may be carefully looked at.
4. Complete the IEEE copyright Form: [Click Here to Download](#).
5. Ensure the figure and table numbers are added in sequential order.
6. Ensure all the technical comments have been incorporated.
7. Acceptance ID mentioned in the acceptance letter.

Important Dates:
Last date of registration: March 22, 2025
Conference Date: 14-16, May 2025

Registration Information available at
<https://icpcsn.com/#registration>

Certificates of Presentation:



Research Paper

A Comparative Analysis of Node.js and Django for Optimizing E-Government Portals: The AYUSH Startup Platform Case Study

Ms. Mrigya Sahai
Student
IT Department
KIET Group of Institutions
sahaimrigya@gmail.com

Ms. Nandini Maheshwari
Student
CSE Department
KIET Group of Institutions
nandinim1204@gmail.com

Mr. Pranjal Raj
Student
CSE Department
KIET Group of Institutions
prayank8c@gmail.com

Ms. Anjali Jain
Assistant Professor
IT Department
KIET Group of Institutions
jainanjali4u@gmail.com

Ms. Mani Dwivedi
Assistant Professor
CSE Department
KIET Group of Institutions
mani.dwivedi@kiet.edu

Abstract - Indian e-Government websites are vital for connecting citizens with public services, yet they often face significant challenges related to accessibility, usability, and performance. Common issues include high latency, inefficient backend systems, poor resource management, and limited adherence to accessibility standards like Web Content Accessibility Guidelines. These obstacles hinder the effectiveness of digital platforms in delivering seamless services. This paper presents a comparative analysis of two popular backend technologies, Node.js and Django, aimed at addressing these challenges, using the proposed Startup AYUSH Portal as a case study. Node.js is known for its scalability, non-blocking architecture, and event-driven model, making it ideal for real-time applications and high-performance systems. On the other hand, Django, a Python-based framework, is renowned for its robustness, security features, and rapid development capabilities, which make it suitable for developing complex, data-driven applications. By examining the limitations of existing e-Government systems, this paper proposes a hybrid solution that leverages the strengths of both Node.js and Django to overcome common backend inefficiencies and enhance user experiences. The study also emphasizes the importance of prioritizing accessibility and performance, ensuring that Indian e-Government platforms meet modern digital demands while being inclusive and efficient.

Index Terms - Indian e-government websites, Web Content Accessibility Guidelines, Node.js, Django, Startup AYUSH Portal, Overcome common backend inefficiencies and enhance user experience

1. INTRODUCTION

As India embraces digital connectivity, e-Government platforms play a vital role in bridging the gap between citizens and services. Despite their growing importance, many Indian e-Government websites face issues like poor accessibility, slow performance, and outdated technologies. These problems, including non-compliance with WCAG 2.0, inefficient backend systems, and high latency, affect user experience, especially in rural areas with limited internet access. This paper examines how Node.js and Django can address these issues, offering scalability, real-time data processing, and rapid development. The proposed Startup AYUSH Portal serves as a case study, aiming to streamline services for AYUSH startups. A hybrid approach combining both technologies can optimize performance, scalability, and accessibility. The research aims to improve Indian e-Government platforms, providing better digital experiences and supporting the vision of Digital India.

2. LITERATURE REVIEW

2.1. Comparative Analysis of Indian Government Websites by using Automated Tool and by End-User Perspective (Authors: Poonam Malik, Dr. Ruchira Bhargava, Dr. Kavita Chaudhary)

The study evaluates Indian e-Government website accessibility with 160 participants and the TAW tool. Major issues were perceivability (524 errors) and robustness (301 errors). MTNL had the most errors (224), while Uttar Pradesh Information Portal had the least (45). Limitations include a small sample, lack of technical assessment, and a single evaluation tool.[1]

2.2. Analysis on Design Issues of E-Government Websites of India (Author: Jatinder Manhas)

The study highlights performance issues in Indian e-government websites, including high latency, unoptimized code, and poor caching. Only 40% meet the recommended page size, and 30% comply with performance standards. It focuses on technical aspects, excluding user experience, causes, and detailed improvement solutions.[2]

2.3. The Indian Government's Web Identity: An Analysis (Authors: Shilpa V, Bijoy P. Joseph)

The study identifies dissatisfaction in Indian government websites, citing design inconsistency, irrelevant media, and task difficulties. It recommends a centralized domain, better servers, and a grievance system. However, the study lacks technical/security analysis and doesn't evaluate the impact of changes on different user groups.[3]

2.4. Government Websites of Kerala: An Evaluation using Government of India Guidelines (Authors: Rajani S., Muralidhara B.L.)

The study evaluates 20 Kerala government websites, revealing poor design, outdated content, and broken links. Most met GII standards, but lacked accessibility features. Recommendations include language support and better archiving. Limitations include predefined metrics, no real-time user feedback, and lack of technical performance analysis.[4]

2.5. Role of Node.js in Modern Web Application Development (Authors: Ghanshyam Jadhav, Flavia Gonsalves)

This paper explores Node.js, a JavaScript runtime using Google's V8 engine and libUV for high performance and concurrency. Its single-threaded event loop enables non-blocking I/O, making it ideal for real-time applications like chat apps and online games. Companies like PayPal use it for scalable web development. While efficient, Node.js struggles with CPU-intensive tasks and has a complex asynchronous model, posing challenges for some developers.[5]

2.6. Comparison and Evaluation of Web Development Technologies in Node.js and Django (Authors: Dr. Anupam Sharma, Archit Jain, Ayush Bahuguna, Deeksha Dinkar)

The study compares Node.js and Django, finding Node.js faster under moderate concurrency, while Django performs better at high concurrency. It excludes security, scalability, and real-world deployment factors, limiting broader applicability.[6]

2.7. Role of Python in Rapid Web Application Development using Django (Authors: Manoj Kumar, Dr. Rainu Nandal)

The study evaluates Django's security, scalability, and integration, highlighting its strengths in rapid development, security protections, and database management. However, it lacks in-depth analysis of advanced features and comparisons with other frameworks like Flask, suggesting future research on additional use cases.[7]

2.8. Efficiency evaluation of Node.js Web Server Frameworks (Author: Danil Demashov and Ilya Gosudarev)

The study tests Node.js web-server frameworks, finding Fastify the fastest, followed by Koa and Restify. Fastify is recommended for speed, Koa for simplicity, and Restify for testing. Limitations include single-core testing, no stress tests, and lack of real-world application analysis.[8]

2.9. Evaluate scalable and high-performance Node.js Application Designs (Authors: Hafiz Umar, Nawaz Denisa Rucaj, and Naveed Kamran)

The study tested scalable Node.js applications, finding a 21% performance boost with multi-instance setups and a 29% increase with multi-machine scaling. However, seven instances caused errors. Limitations included a single database bottleneck, hardware constraints, and JMeter memory issues, limiting broader scalability testing.[9]

2.10. Node.js Challenges in Implementation (Authors: Hezbullah Shah, Tariq Rahim Soomro)

This study combines a literature review and a survey of 80 developers to analyze Node.js adoption. It highlights benefits like full-stack JavaScript development, cost reduction, and suitability for SPAs and high-load tasks. However, challenges include a steep learning curve, complex asynchronous features, and resistance to replacing established technologies. Security risks, organizational hesitance, and low market awareness further limit adoption, despite Node.js's potential advantages.[10]

2.11. SODAR Core: a Django-based framework for scientific data management and analysis web apps (Authors: Mikko Nieminen, Oliver Stolpe, Franziska Schumann, Manuel Holtgrewe, and Dieter Beule)

The study presents SODAR Core, a modular framework for managing scientific datasets under FAIR principles, featuring project-based access, asynchronous jobs, and caching. Limitations include fixed role access control and consistency challenges, with future improvements aimed at enhancing flexibility.[11]

2.12. Review Paper on Django Web Development (Author: Tokpam Sushilkumar Singh, Harmandeep Kaur)

This paper explores Django, a Python-based web framework using the MVT structure. It covers installation, project creation, ORM-based database interaction, and CRUD operations. The study highlights Django's speed, scalability, and beginner-friendly nature but lacks advanced insights on performance optimization, robust databases, and scaling strategies, making it more suited for beginners than experienced developers.[12]

2.13. Indian Government Websites: A Study (Authors: B.B. Chand and Ramesha)

The study evaluates 81 Indian central government websites based on content, design, accessibility, and engagement. While some perform well, many face usability and interactivity issues. It recommends improvements for better public service delivery. Limitations include the exclusion of state-level sites, lack of security analysis, and a framework that may not fully capture user experience.[13]

2.14. An Approach for Automated Code Deployment Between Multiple Node.js Microservices (Authors: Oleh Chaplia, Halyna Klym)

The paper presents an automated deployment framework for Node.js microservices, integrating with CI/CD pipelines to ensure consistency, reliability, and minimal downtime. The framework reduces human errors and accelerates updates. However, it focuses solely on Node.js, lacks analysis of rollback scenarios, and may require significant setup for integration with existing pipelines.[14]

2.15. Service-Oriented Government Websites Construction Research (Authors: Ran Tang, Zhenji Zhang, Yue Yin)

The paper proposes a framework for service-oriented government websites, emphasizing usability, reliability, and clear information. It highlights how prioritizing user needs improves public service delivery. However, the study is specific to China, lacks cybersecurity considerations, and does not provide detailed

implementation strategies or case studies to validate its effectiveness.[15]

3. METHODOLOGY

The methodology aims to evaluate the current state of Indian government websites, the impact of backend technologies on user experience, and propose solutions using modern technologies like Node.js and Django.

3.1. Comparative Methodology:

This aspect involves comparing the current state of Indian government websites with global standards in terms of usability, accessibility, and performance. By looking at existing research on **Node.js** and **Django**, compare how these technologies can potentially improve the backend infrastructure of government websites when compared to the current technologies they use.

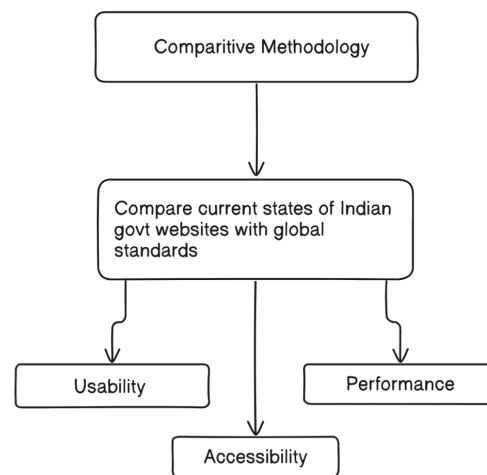


Figure 1: Comparative Methodology to compare current states of different e-gov websites

3.2. Analytical Methodology:

This involves an analysis of the problems identified in the existing systems of Indian government websites (such as slow performance, poor scalability, and accessibility issues). Analyze the performance data of these sites and propose **Node.js** and **Django** as potential solutions. The analytical approach also includes evaluating the **backend performance**, **latency**, **resource optimization**, and **usability** improvements that could result from implementing these modern technologies.

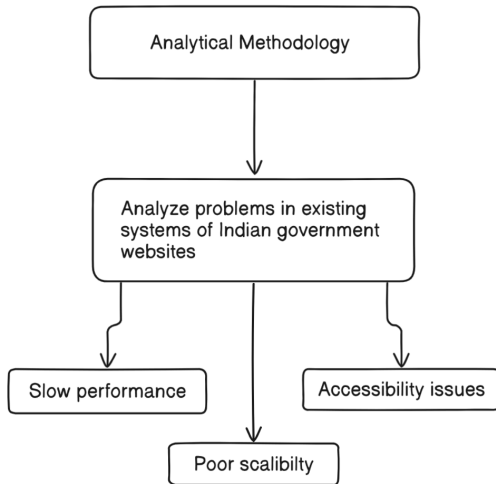


Figure 2: Analytical Methodology to analyze problems in existing systems

3.3 Step-by-Step Approach:

Problem Identification:

Key issues in Indian government websites include poor performance, usability challenges, and accessibility limitations. Previous studies highlight the need for modern technologies to address these problems.

Comparative Analysis:

The study compares Indian government websites with global usability and accessibility standards. It evaluates backend performance and suggests adopting Node.js and Django for better scalability, latency, and resource management.

Technological Solutions Exploration:

- **Node.js:** Enhances backend performance with non-blocking, event-driven architecture, improving response times and resource handling.
- **Django:** Supports real-time data sync and cloud-based infrastructure to improve scalability and performance consistency.

Implementation of Technical Solutions:

The research explores how integrating Node.js and Django can enhance real-time interactions, scalability, and overall website performance.

Evaluation and Impact Assessment:

The study assesses improvements in latency, resource optimization, scalability, and user experience after implementing these technologies. For evaluation we can

utilize A/B testing approaches for user experience improvement.

Conclusion and Recommendations:

Findings suggest that Node.js and Django can significantly enhance government website performance. The research proposes a roadmap for their integration to improve scalability and usability.

4. IMPLEMENTATION AND RESULTS

PROPOSED FEATURES:

Real-time User Dashboard: The real-time user dashboard is designed to provide immediate updates and interaction capabilities for users on the AYUSH Startup Portal. The dashboard includes features like user activity tracking, notifications, and analytics, offering a dynamic and engaging interface. Real-time communication is enabled through WebSockets, allowing the dashboard to push updates to connected users without requiring them to refresh the page.

Purpose: To allow users to view dynamic data such as system notifications, analytics, and real-time updates without delays. This ensures better engagement and user experience.

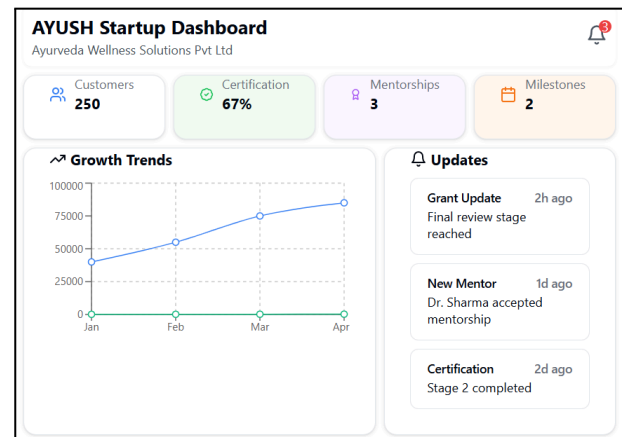


Figure 3: UI for Real time dashboard

Startup Registration and Verification: This feature allows AYUSH startups to register on the platform by providing essential details such as name, address, and supporting documents for verification. It ensures secure handling of user information and offers a guided form submission process. Verification mechanisms are incorporated to validate the authenticity of startups.

Purpose: To simplify the onboarding process for AYUSH startups while maintaining high security and compliance standards.

Startup Registration

Startup Name

Enter startup name

Email

Enter email

Phone

Enter phone number

Address

Enter complete address

Supporting Documents

Choose File

No file chosen

Register

Figure 4: UI for Startup registration and verification

Search and Filter for Startups: The search and filter feature enables users to find AYUSH startups by applying various criteria, such as location, industry type, funding status, or keywords. The feature supports advanced searching capabilities, including full-text search and filtering options, ensuring users can efficiently access relevant information.

Purpose: To enhance the usability of the portal by making it easier for users to find specific startups from a potentially large dataset.

Yoga startups in Bangalore

Example searches:

Yoga startups in Bangalore

Funded Ayurveda companies

Homeopathy startups Series A

Location

Industry

Funding

Figure 5: UI for search query

Search startups...

Location

Industry

Funding

Location

Delhi

Mumbai

Bangalore

Chennai

Hyderabad

Figure 6: UI for filtering location

Search startups...

Location

Industry

Funding

Industry

Ayurveda

Yoga

Unani

Siddha

Homeopathy

Figure 7: UI for filtering industry

Search startups...

Location

Industry

Funding

Funding

Bootstrapped

Seed

Series A

Series B+

Figure 8: UI for filtering funding

API Integration for External Data: This feature integrates external data sources to enhance the functionality of the AYUSH Startup Portal. APIs from government or other third-party organizations provide data on AYUSH schemes, funding opportunities, or regulations. The integration ensures the portal is comprehensive and keeps users informed about external opportunities and resources.

Purpose: To provide startups with access to additional, reliable resources directly through the platform, saving them time and effort in manual research.

Implementation and Results Table		
Feature	Implementation	Results
Real-time User Dashboard	Node.js: Utilized its non-blocking , event-driven architecture (to reduce latency) with Socket.IO for real-time updates.	High concurrency handling with consistent response time, even under load. Real-time updates were seamless for up to 200 concurrent users.[6]
	Django: Used WebSockets with Django Channels for real-time updates.	Achieved functional real-time updates but faced latency when handling more than 100 concurrent users, leading to slower response times.
Startup	Node.js: Used	Faster performance

Registration and Verification	Express.js with Multer for file uploads and bcrypt for secure authentication.	but required additional effort to implement security features, such as CSRF protection, manually.
	Django: Employed its built-in ORM for secure database transactions and integrated Django Rest Framework (DRF) for the API layer.	High security during data upload and verification process. Easy implementation of form validation and secure file handling with Django's robust middleware.
Search and Filter for Startups	Node.js: Utilized MongoDB with Mongoose for query handling and Elasticsearch for advanced search functionality.	Faster response times for complex queries and large datasets, leveraging asynchronous database operations and indexing capabilities of Elasticsearch.
	Django: Implemented full-text search using PostgreSQL and Django ORM query optimizations (to improve performance)	Efficient query handling for moderate dataset sizes. Experienced latency under high data loads due to synchronous nature of Django ORM.
API Integration for External Data	Node.js: Used Axios for API requests and processed responses asynchronously.	Handled large datasets efficiently with asynchronous calls. Scalability for future API integrations was seamless.
	Django: Leveraged DRF for API handling with pre-built serializers to process incoming data.	Quick integration with pre-built tools but slower response times compared to Node.js under high data loads.

Table 1: Feature Implementation and their reasoning

5. FUTURE SCOPE

5.1. Cloud Integration and Microservices Evolution

The future scope of research in Indian government websites presents numerous opportunities for technological advancement and optimization. A primary area of exploration lies in **Cloud Integration** and **Microservices Architecture**, where research could focus on migrating existing government infrastructure to cloud-based solutions.

5.2. Performance Optimization Frameworks

A critical avenue for future research involves developing comprehensive **Performance Optimization Frameworks**. This research direction would encompass creating standardized benchmarks specifically tailored for government websites, incorporating **machine learning algorithms** for predictive performance analysis, and implementing **automated load balancing** (to achieve scalability) strategies.

5.3. Advanced Security Protocols

The realm of **Security Enhancement** presents another crucial research direction. Future studies could explore the implementation of **blockchain technology** for secure document verification and the adoption of **zero-trust architecture** in government web applications. This research would be particularly valuable given the sensitive nature of government data and the increasing sophistication of cyber threats.

5.4. Enhanced Accessibility and User Experience

Accessibility and User Experience represent a significant area for future investigation. Research could explore the implementation of **AI-powered accessibility features** and **Progressive Web Apps (PWAs)** in government websites. A particular focus could be placed on developing **multilingual support optimization** using modern backend technologies, which is crucial in a diverse country like India.

5.5. Advanced Analytics and Reporting Systems

Data Analytics and Reporting present another crucial area for future research. This involves studying **real-time analytics** implementation for government service usage and exploring **big data processing** capabilities using Node.js and Django. The research could focus on developing **predictive analytics** models to optimize service delivery and resource allocation.

5.6. Cross-Platform Compatibility Solutions

The scope of **Cross-Platform Compatibility** research extends to developing **universal design patterns** that ensure consistent performance across different browsers and devices. This includes studying **progressive enhancement techniques** for varying internet speeds and investigating solutions for maintaining compatibility with legacy systems, which is particularly relevant in the government sector.

5.7. API Standardization and Integration

Future research in **API Standardization** could focus on developing standardized patterns for government services, implementing robust **API gateway** solutions, and exploring **GraphQL integration** for optimized data fetching. This research direction is crucial for creating interoperable government services that can efficiently share data while maintaining security standards.

5.8. Advanced Monitoring and Testing Frameworks

Lastly, research in **Performance Monitoring and Testing** could explore the development of automated testing frameworks specifically designed for government websites. This includes studying **real-time monitoring solutions** for early problem detection and investigating specialized **load testing methodologies** that account for the unique characteristics of government portal usage patterns.

6. CONCLUSION

Real-time Features: Node.js is the preferred choice for real-time functionalities like the user dashboard. Its event-driven architecture provides a clear advantage in handling concurrent users efficiently, making it ideal for scenarios with high traffic and dynamic updates.

Secure and Structured Applications: For features requiring a secure, structured approach, such as startup registration and verification, Django offers a robust framework with pre-built security measures. Its ORM simplifies database interactions while maintaining high standards of data integrity and security.

Data-Intensive Applications: Node.js is better suited for tasks involving large datasets, complex queries, or API integrations. Its asynchronous processing and support for tools like Elasticsearch enable it to handle such tasks with higher efficiency.

Hybrid Approach: The combination of both technologies can be leveraged for an optimal solution. For instance, Django can manage the secure and

structured backend for sensitive operations, while Node.js handles real-time updates and data-intensive tasks.

7. REFERENCES

- [1]Malik, P., Bhargava, R., & Chaudhary, K. (2017). "Comparative Analysis of Indian Government Websites by using Automated Tool and by End-User Perceptive". *International Journal of Allied Practice, Research and Review*
- [2]Manhas, J. (2014). "Analysis on design issues of E-government websites of India". *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(2), 646-650.
- [3]Shilpa, V., & Joseph, B. P. (2024). "The Indian Government's Web Identity: An Analysis". *European Journal of Arts, Humanities and Social Sciences*, 1(4), 67-74.
- [4]Rajani, S., & Muralidhara, B. (2016). "Government websites of kerala: an evaluation using government of India guidelines". *International Journal of Computer Applications*, 975, 8887.
- [5]Jadhav, G., & Gonsalves, F. (2020). "Role of Node. js in Modern Web Application Development". *Int. Res. J. Eng. Technol*, 7(6), 6145-6150.
- [6]Sharma, D. A., Jain, A., Bahuguna, A., & Dinkar, D. (2019). "Comparison and Evaluation of Web Development Technologies in Node. js and Django". *Int. J. Sci. Res*, 9(12), 1416-1420.
- [7]Kumar, M., & Nandal, R. (2024). "Role of Python in Rapid Web Application Development Using Django". *Available at SSRN 4751833*.
- [8]Demashov, D., & Gosudarev, I. (2019). "Efficiency Evaluation of Node. js Web-Server Frameworks". In *MICSECS*.
- [9]Nawaz, H. U., Rucaj, D., & Kamran (2018), N. "Evaluate scalable and high-performance Node. js Application Designs".
- [10]Shah, H., & Soomro, T. R. (2017). "Node. js challenges in implementation". *Global Journal of Computer Science and Technology*, 17(2), 73-83.
- [11]Nieminen, M., Stolpe, O., Schumann, F., Holtgrewe, M., & Beule, D. (2020). "SODAR Core: a

Django-based framework for scientific data management and analysis web apps". *Journal of Open Source Software*, 5(55), 1584.

[12]Singh, T. S., & Kaur, H. (2023). "Review Paper on Django Web Development". *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH*.

[13]Chand, B. B., & Ramesha, B. (2017). "Indian Government Websites: A Study". *DESIDOC Journal of Library & Information Technology*, 37(5), 346.

[14]Chaplia, O., & Klym, H. (2023, September). "An Approach for Automated Code Deployment Between Multiple Node. js Microservices". In *2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT)* (pp. 202-205). IEEE.

[15]Tang, R., Zhang, Z., & Yin, Y. (2011). "Service-oriented Government Websites Construction Research". In *ICEIS (I)* (pp. 550-553).

15%

SIMILARITY INDEX

13%

INTERNET SOURCES

6%

PUBLICATIONS

9%

STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to KIET Group of Institutions, Ghaziabad Student Paper	1%
2	Submitted to Delhi Metropolitan Education Student Paper	1%
3	www.ijcaonline.org Internet Source	1%
4	www.ijaprr.com Internet Source	1%
5	joiv.org Internet Source	1%
6	ejahss.com Internet Source	1%
7	www.coursehero.com Internet Source	1%
8	www.igi-global.com Internet Source	1%
9	pure.mpg.de Internet Source	1%
10	hj.diva-portal.org Internet Source	<1%
11	Submitted to National College of Ireland Student Paper	<1%
12	www.scitepress.org Internet Source	<1%
13	pdfs.semanticscholar.org Internet Source	<1%